gnuplot 3.5 User's Guide

K C Ang
Division of Mathematics,
NTU-NIE, Singapore.
kcang@nie.edu.sg

June 2000

1 Introduction

This document introduces the new user to `gnuplot` for Windows. It is a brief document to get the reader started on using `gnuplot` to produce a variety of graphs. It is not meant to be a manual since a detailed online help manual comes with the software.

`gnuplot` is an interactive plotting program. The two main plotting commands are `plot` and `splot`. The other commands sets various parameters and give you options to customise your plots.

Besides using `gnuplot` as an interactive software, one also write *script files* and load the file from inside `gnuplot`. Alternatively, one can run the script in *batch mode* from the DOS prompt.

Some of the things that `gnuplot` can do include:

- Simple plots of built-in or user-defined functions
- Scatter plots of bivariate data, with errorbar options
- Bar graphs
- Three-dimensional Surface plots of functions of the form $z = f(x, y)$ with options for hidden line removal, view angles and contour lines
- Two- and three-dimensional plots of parametric functions
- Plots of data directly from tables created by other applications
- Re-generate plots on various output files or devices

An example of a plot created in `gnuplot` is shown in Figure 1.

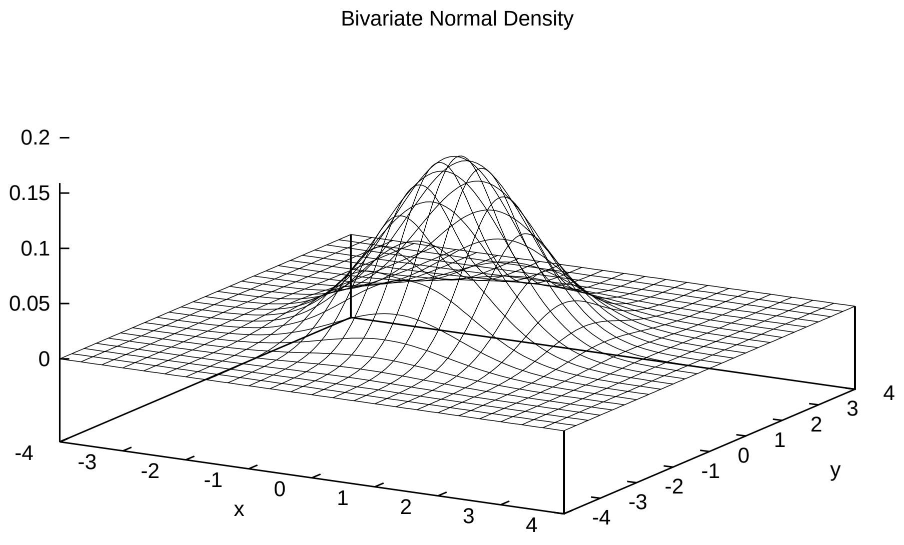


Figure 1: An example of a plot produced by `gnuplot`

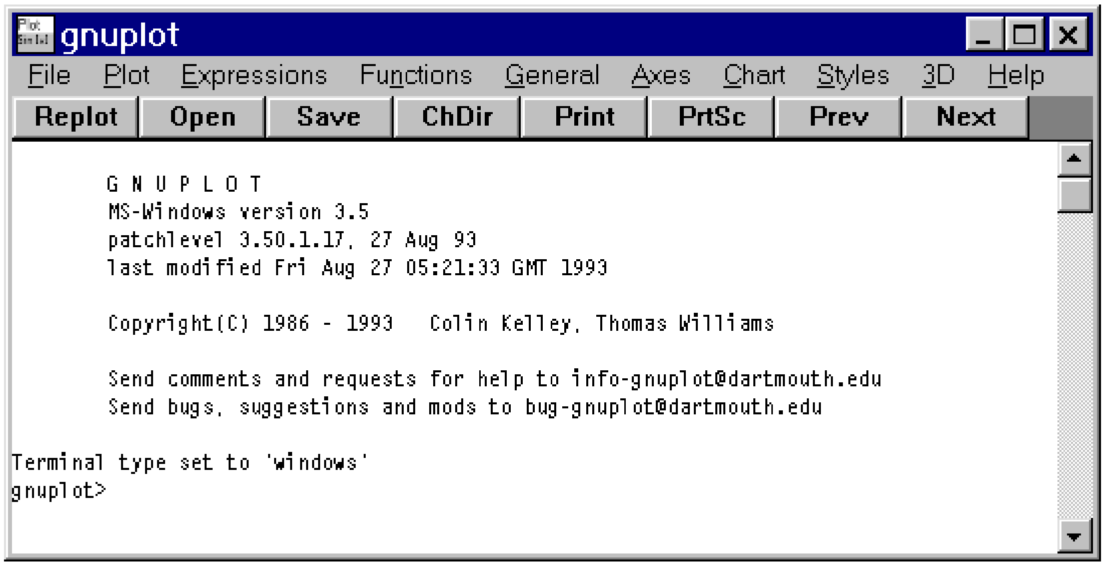
2 Starting gnuplot

gnuplot is available on computers running a variety of operating systems, including IBM-PCs and compatibles, UNIX, VAX/VMS, DOS and Windows9x. In this document, it is assumed that the reader uses a PC (running MS Windows95/98). It is also assumed that gnuplot has been properly installed on the system.

To start gnuplot, one simply double-clicks on the gnuplot icon:

wgnuplot

The following gnuplot window should pop up:


gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D Help
Replot Open Save ChDir Print PrtSc Prev Next

```
       G N U P L O T
       MS-Windows version 3.5
       patchlevel 3.50.1.17, 27 Aug 93
       last modified Fri Aug 27 05:21:33 GMT 1993

       Copyright(C) 1986 - 1993   Colin Kelley, Thomas Williams

       Send comments and requests for help to info-gnuplot@dartmouth.edu
       Send bugs, suggestions and mods to bug-gnuplot@dartmouth.edu

Terminal type set to 'windows'
gnuplot>
```

The menu bar, tool bar and buttons that appear on the window look like those on any other Windows-based applications. This is the *interactive* gnuplot environment.

To exit gnuplot, you can type either exit, quit or simply q. Alternatively, you can exit by clicking on File-Exit on the menu bar.

Online Help is available by either typing help at the gnuplot command prompt or clicking the Help menu item.

gnuplot has a feature that allows you to recall previous commands and edit them. On the PC, the up/down arrow keys are used to get the previous/next commands.

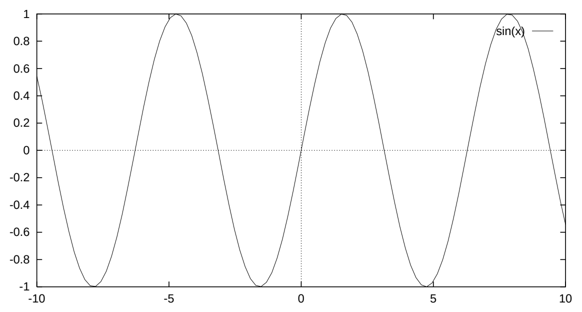
3 Basics

3.1 Plotting pre-defined functions

Start exploring `gnuplot` by typing the following commands at the `gnuplot` prompt:

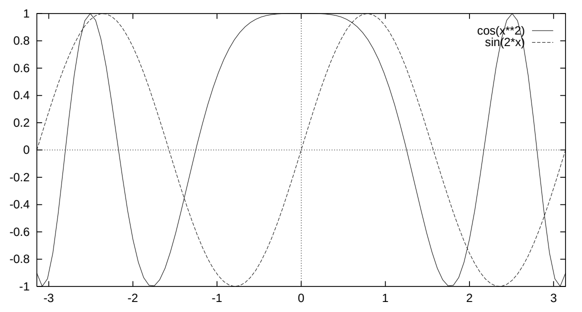
1. `plot sin(x)`

   This command produces a plot of $\sin x$:



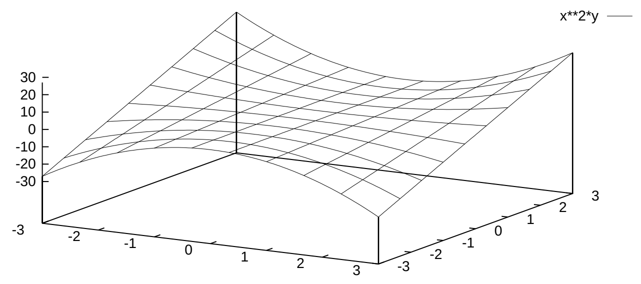
2. `plot [-pi:pi] cos(x**2), sin(2*x)`

   This command creates a plot of $\cos x^2$ and $\sin 2x$ on the same graph, with x ranging from $-\pi$ to π:



3. `splot [-3:3] [-3:3] x**2*y`

   This command produces a three-dimensional surface plot of the function $f(x, y) = x^2y$:



In the above three examples, we have used `gnuplot`'s built-in functions `sin()`, `cos()` and `exp()`. We have also used the built-in or pre-defined constant `pi`. The complete list can be found in `gnuplot`'s manual.

In the first example above, the `plot` command tells `gnuplot` to produce a two-dimensional plot. As the range for x is not specified, `gnuplot` uses its default range of $(-10, 10)$.

In the second example, we specify the range for x (from $-\pi$ to π) by typing `[-pi:pi]` following the `plot` command. The two functions to be plotted are separated by a comma. Note that on a colour screen, `gnuplot` uses different colours for the functions. On a monochrome display (or if the output is directed to a non-colour eps file as in the case here), the functions are plotted using different line styles.

In the last example, we set the x and y values to range from -3 to 3. Note that the ranges are optional in both `plot` and `splot` commands.

A sample of `gnuplot`'s pre-defined functions is given below:

`gnuplot` function	Description
`abs`	absolute value
`acos`	inverse cosine
`asin`	inverse sine
`atan`	inverse tangent
`cos`	cosine
`cosh`	hyperbolic cosine
`exp`	exponential
`floor`	largest integer not greater than its argument
`int`	integer part of the argument
`log`	natural logarithm
`log10`	common logarithm (i.e. base 10)
`rand`	pseudo random number
`sgn`	sign function
	(1 if argument is positive, -1 if negative, 0 if zero)
`sin`	sine
`sinh`	hyperbolic sine
`sqrt`	square root
`tan`	tangent
`tanh`	hyperbolic tangent

For a complete list, refer to the online help on `gnuplot`.

3.2 More on Setting Ranges

If you wish to specify the y-range but not the x-range for a plot, put in both sets of brackets but leave the first empty. For instance,

```
plot [] [0:4] 1/(x**3)
```

The command `set` has many options and is used to control various components of a plot. In the previous examples, we saw how we can set the ranges of our plots by specifying them in the `plot` or `splot` commands. The ranges can also be set before plotting using the commands `set xrange`, `set yrange` and `set zrange`. For instance,

```
set xrange [-3*pi:3*pi]
set yrange [:20]
set zrange [-exp(2.7):sin(pi/2)]
```

You may omit either the upper or the lower limits. The limits may be numbers or pre-defined constants, or expressions.

3.3 Setting Labels and Titles

We may give the axes a label and the plot a title using commands such as the following:

```
set xlabel 'x'
set ylabel "Host Population"
set title 'Graph of y against x'
replot
```

Note that both single quote and double quote are acceptable. The `replot` command simply redraws the previous plot, incorporating new settings or changes.

4 User-defined Constants and Functions

4.1 Simple constants and expressions

If we use constants or functions repeatedly, we may wish to define them using names that we can remember easily. For this purpose, we will need to know how arithmetic and expressions are written in `gnuplot`. The arithmetic and logical expressions used in `gnuplot` are basically similar to those used in Fortran or C.

Some common binary operators are listed below:

Symbol	Example	Explanation
`**`	`a**b`	exponentiation
`*`	`a*b`	multiplication
`/`	`a/b`	division
`%`	`a%b`	modulo
`+`	`a+b`	addition
`-`	`a-b`	subtraction
`==`	`a==b`	equality
`!=`	`a!=b`	inequality

As an example of a user-defined function with some fixed constants, suppose we wish to graph the function

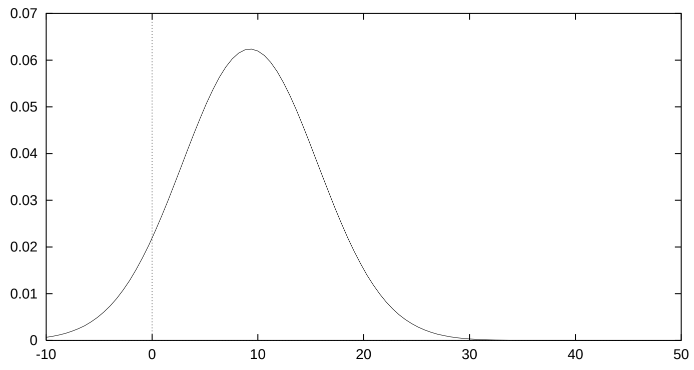
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where $\mu = 9.256$ and $\sigma = 6.395$.

The following set of commands may be used:

```
m=9.256
s=6.395
f(x) = 1/(sqrt(2*pi)*s)*exp(-(x-m)**2/(2*s**2))
plot [-10:50] f(x)
```

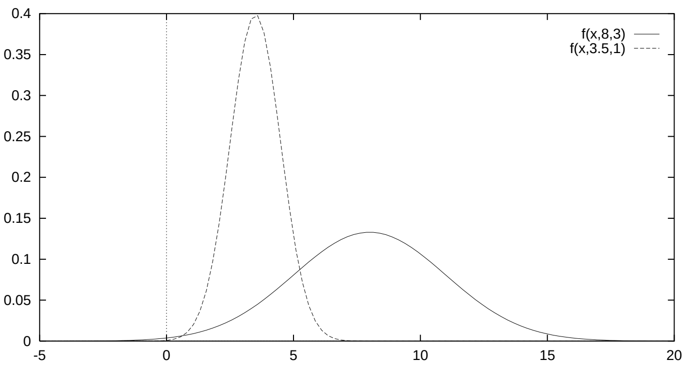
This produces the following graph:



Suppose we wish to the different plots produced by different sets of values of μ and σ, we may do the following:

```
f(x,m,s) = 1/(sqrt(2*pi)*s)*exp(-(x-m)**2/(2*s**2))
plot [-5:20] f(x,8,3), f(x,3.5,1)
```

The following is obtained:



4.2 Piecewise continuous functions

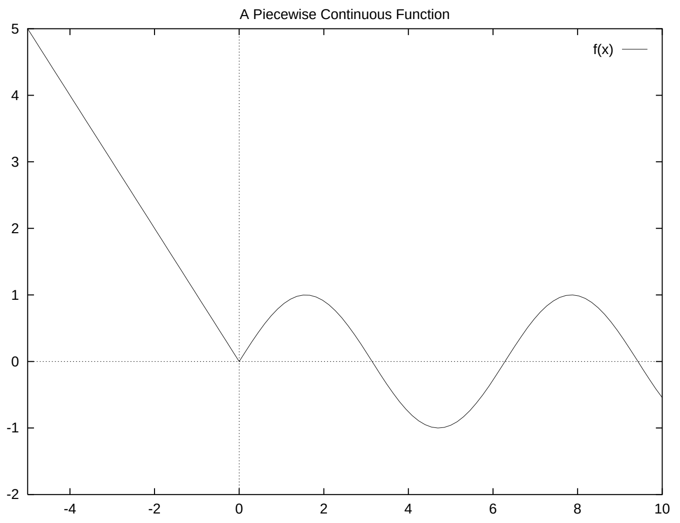
`gnuplot` can also plot functions which are piecewise continuous. For instance, if we wish to plot the following function

$$f(x) = \begin{cases} -x, & x < 0 \\ \sin(x), & x \geq 0 \end{cases} .$$

We can use the commands:

```
f(x)=(x<0) ?  -x:sin(x)
plot [-5:]  [-2:]  f(x)
```

and obtain the following graph:



In the above example, we have used `gnuplot`'s ternary operator:

`<condition> ? <expression1> : <expression2>`

which means if <`condition`> is true, then <`expression1`> is evaluated, otherwise <`expression2`> is evaluated.

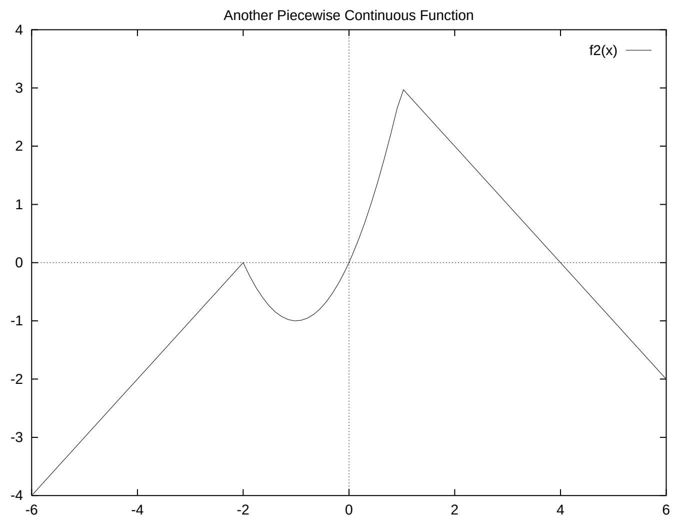
In order to define continuous piecewise functions which have more than two pieces, we need to use the following trick: suppose we have a function defined as

$$f(x) = \begin{cases} x+2, & x < -2 \\ x^2+2x, & -2 \le x < 1 \\ 4-x, & x \ge 1 \end{cases} .$$

In order to define this function in `gnuplot`, we string two functions together and then plot the second function:

```
f1(x)=(x<-2) ?  x+2 :  x**2+2*x
f2(x)=(x<1) ?  f1(x) :  4-x
plot [-6:]  [:4] f2(x)
```

The following graph results:



Note that `gnuplot` will connect the endpoints with straight lines even if the piecewise function is not continuous. This is not satisfactory as we may wish to plot discontinuous functions at times. One way to overcome this is to set the unwanted sections to something unprintable. For instance, try the following:

```
f1(x)=(x<0) ?  -1 :  sqrt(-1)
f2(x)=(x<0) ?  x/0 :  1
plot [] [-2:2] f1(x), f2(x)
```

5 Script files and Batch Processing

It is quite common to find ourselves using the same set of commands over and over again. `gnuplot` allows us to put such commands in a *script file*. The file simply consists of the commands or statements that you would enter interactively at the `gnuplot` prompt. We can use a text editor (such as Notepad) to create or edit such a file.

Once such a script file is created, it can be run or processed in two ways. Suppose the script file is called "mygraph.gp". First, you can run `gnuplot` and at the `gnuplot` prompt, type:

```
load "mygraph.gp"
```

Alternatively, you can run the script file at the DOS prompt by typing:

```
wgnuplot mygraph.gp
```

This will invoke `gnuplot`. After it finishes executing the commands in the script, it exits. Note that there is a `pause` command which can be used to pause the execution of the commands in the script until the user presses the `RETURN` key.

The script file used to produce the previous plot is given here:

```
# script for plotting a piecewise continuous function
set terminal postscript eps 14
set size 1.0, 1.0
set output "graph.eps"        # save output to a file
set title "Another Piecewise Continuous Function"
f1(x)=(x<-2) ? x+2 : x**2+2*x
f2(x)=(x<1) ? f1(x) : 4-x
plot [-6:] [:4] f2(x)
```

Note that the first line begins with a "`#`". Anything that comes after this symbol is a comment and will be ignored by `gnuplot`. Hence the first line is simply a comment. Note also that on line 4, we place a comment after the command "set output ...". This is done to remind ourselves what we are doing, making the script more readable.

If a command is too long and needs to be continued to the next line, put a backslash, \, at the end of the line to be continued. Note that the \ character must be the last character in the line (no spaces should come after it). Multiple commands can be placed on a single line by separating them with a semicolon, ;.

Suppose we have started an interactive session of `gnuplot`. Suppose we have defined some variables and functions and customised some settings. If we wish to keep all these so that we can use it again later, we can write them (along with all of the other default values and commands) to a file by giving the command:

```
save 'myfile.gp'
```

6 Plotting Data files

gnuplot can produce plots from tabulated data. This is very useful when we have data obtained from another application and we wish to produce a graph for the set of data. Note that in the data file, we can put comments in the same way as we can in a script file; gnuplot ignores everything that comes after the # symbol.

In the discussion to follow, we shall assume that our data sets are stored in a file called "values.dat". The file should contain one data point per line. For example, for a one variable data set, the file may contain the following set of data for that variable:

```
# a sample data set
 2.2
 3.5
 5.6
 8.9
 7.4
```

Using the command:

```
plot 'values.dat',
```

we can plot the points (0,2.2), (1,3.5), (2,5.6), ... That is, the data set gives the y-coordinate. The corresponding x-coordinate starts from 0 and increases by 1 for each value.

Suppose we have a data set consisting of (x, y) pairs. The data set should be organised in columns as in the following example:

```
# another sample data set
0.5   2.2
1.2   3.5
1.8   5.6
3.2   8.9
4.9   7.4
```

Suppose we wish to plot these points and join them using straight lines. We may use either one of the following commands:

```
plot 'values.dat' with lines
plot 'values.dat' with linespoints
```

The option that comes after `with` is called a **plot style**. The `lines` style connects adjacent points with lines. The `linespoints` style connects the points as well as displays a symbol at the points. For more information on plot styles, consult the online help by typing

```
help plot styles
```

at the gnuplot prompt.

Suppose we have a table of values where only the first column is the independent variable and the other columns contain values of some other dependent variables. For instance, one data set could look like this:

```
# a table of values
# t     P      Q      R
  0    1.2    2.2    3.8
  1    2.1    2.9    3.6
  2    2.9    3.1    2.2
  3    3.5    3.7    1.1
  4    4.0    4.5    0.2
```

To obtain a plot of P, Q and R against t on the same pair of axes, we could, for example, use the command:

```
     plot 'values.dat' title 'P' with linespoints, \
          'values.dat' using 3 title 'Q' with linespoints, \
          'values.dat' using 4 title 'R' with linespoints
```

Recall that the backslash represents a continuation to the next line. In the above command, we have given a title to each plot. These will appear in the **key** or legend of the plot. The number following the word `using` tells `gnuplot` the column to find the values to be plotted.

As a further example, consider the following set of data obtained from a simulation of an epidemic. The values were obtained from a Fortran program, tabulated and saved to a file called “epi.dat”. A portion of the file is shown below:

```
  1    979.510     1.440    20.050
  2    978.805     2.073    20.122
  3    977.790     2.984    20.226
  4    976.331     4.294    20.375
...    .......    ......   .......
...    .......    ......   .......
 67      0.038    83.113   917.849
 68      0.036    78.959   922.005
 69      0.035    75.013   925.953
 70      0.033    71.263   929.703
```

We could use the following script to create a plot:

```
   set terminal postscript eps 14
   set size 1.2, 1.0
   set output "epi.eps"
   set key 60,500
   plot "epi.dat" title 'x(n)' with linespoints, \
        "epi.dat" using 3 title 'y(n)' with linespoints, \
        "epi.dat" using 4 title 'z(n)' with linespoints
```

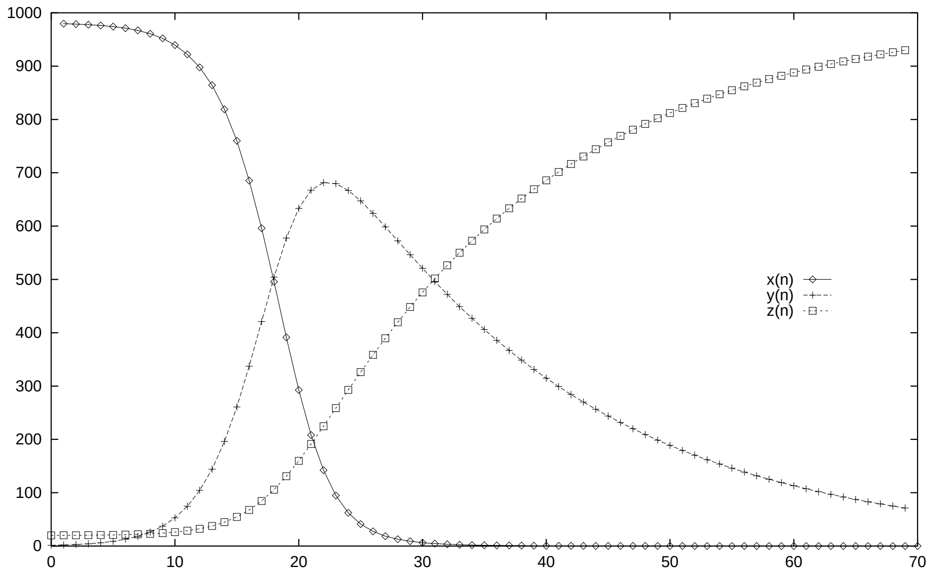
In the above script, we first set the terminal to `postscript eps` so as to create an eps (encapsulated postscript) file. the number "14" refers to the font size of any text in the file.

The `set size` command basically tells `gnuplot` the relative dimensions for the width and height (respectively) of the plot. In this example, we want the plot to be 1.2 units wide and 1.0 units high.

The `set output` command sets the output to the specified device. In this case, we wish to have the plot saved to a file called "epi.eps" so that we can include it as a figure in, say, a LaTeX document.

`set key x, y` places the key (i.e. legend) of the plot at position (x, y) on the plot.

The script produces the following plot (saved in the file "epi.eps"):



There are many other ways of customising plots using the `set` command. For instance, we may add text to the plot (annotate the graph) using the `set label` option. We can also draw arrows or line segments using the `set arrow` command. Tick marks on the axes can also be set using the `set xtics` command for instance.

For more information on the `plot` and `set` commands, as well as `using` and `with` options associated with the `plot` or `splot` commands, one can refer to the online help manual that comes with `gnuplot`.

7 Output of Plots

After creating a plot, we usually wish to be able to obtain a hard copy of it, or perhaps to put the plot onto another document. This can be done in a variety of ways.

One easy way to create a document containing the plot is to copy it to the clipboard and then paste it onto a word processing document, like MS WORD, for instance. Suppose we have created a plot using `gnuplot` and a window displaying the resulting plot pops up. We can then *right-click* anywhere on the plot and a menu will pop up. Select `Copy to Clipboard` from the menu items and the plot will be copied. We may now open a new WORD document and paste the plot there.

If we wish to output the plot to some other graphic files, we can use the `set terminal` command. We have already seen how to create an encapsulated postscript (eps) file earlier. The eps file created may then be included in other documents (for example, in a LaTeX source document).

Another way to created a plot to be included in a LaTeX document is to actually create the LaTeX code for the plot instead of producing an eps file. This is done by using the setting the terminal option to `latex`. Setting the terminal type to latex tells `gnuplot` that we wish to create a plot in LaTeX's picture environment format. `gnuplot` will create the necessary plot and then convert to LaTeX's graphics language.

Below is an example of a script file used to produce a LaTeX file for the plot:

```
set terminal latex
set output 'plot.tex'
set title 'An Example of plotting in \LaTeX\ with {\sf gnuplot}'
set xrange [-pi:pi]
plot sin(x) title '$\sin x$', cos(x) title '$\cos x$'
```

Note that in this script, we have included some LaTeX code in the `gnuplot` commands. For instance, the titles for the individual plots are written in LaTeX's mathematics environment, hence the pair of `$` signs.

Running the above script creates a file called "plot.tex", which contains the LaTeX source code for the plot. This file can be inserted into the main LaTeX document. To do so, at the point in the document where the plot is to appear, we insert the following:

```
\begin{figure}
\begin{center}
\input{plot.tex}
\end{center}
\caption{An Example of a plot generated with the {\tt latex} terminal type}
\end{figure}
```

The following plot is obtained:

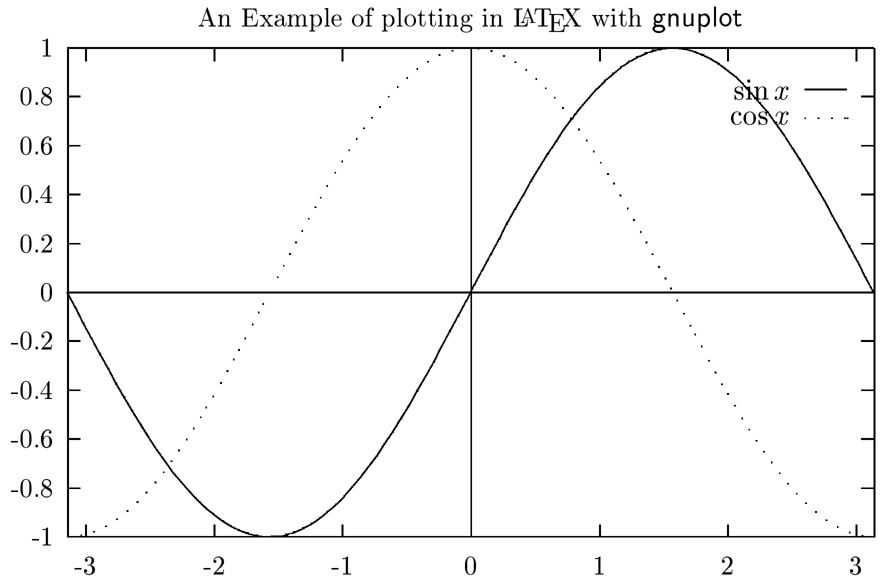


Figure 2: An Example of a plot generated with the `latex` terminal type